RESAD: ROUND-EFFICIENT SECURE AUTOREGRESSIVE DECODING SCHEME FOR TRANSFORMER INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Secure Transformer inference for generative large language models can provide protection for both user inputs and model parameters. Existing secure transformer inference schemes based on secure multi-party computation suffer from high latency during secure autoregressive decoding. This latency is amplified by sequential token generation: each token requires a secure inference pass, and the communication rounds of secure protocols are accumulated over many decoding steps. In this paper, we propose RESAD, a round-efficient secure autoregressive decoding scheme for Transformer-based models. To alleviate the performance bottlenecks introduced by nonlinear functions in secure decoding, we design low-round secure computation protocols, with a focus on Softmax and GeLU. For Softmax, a novel bucket-based maximum approximation algorithm is proposed to reduces secure maximum computation from logarithmic-round to constant-round. We further introduce a novel interval lookup-table-based approximation protocol which is used not only for secure evaluation on the exponential and reciprocal functions in few rounds, but also for the secure evaluation of GeLU and LayerNorm. These techniques substantially reduce the interaction overhead of nonlinear functions in secure autoregressive decoding. These techniques are integrated into an end-to-end secure Transformer inference scheme optimized for autoregressive decoding. Experiments on Transformer models show that, compared with the state-of-the-art efficient scheme, RESAD reduces communication rounds by 56.4%-63.2%, online communication by 28.9%-51.4% and end-to-end runtime by 24.4%-43.3% in WAN settings and by 14.5%-16.4% when input 1024th tokens under LAN, while maintaining comparable accuracy. We further implement and evaluate RESAD on TinyLlama, demonstrating its applicability to LLaMA-style models.

1 INTRODUCTION

Generative large language models (LLMs) based on the Transformer (Vaswani et al., 2017) have been widely deployed in the Machine Learning as a Service (MLaaS) paradigm to support applications such as code generation and question answering. However, this deployment model raises serious privacy and security concerns. On the one hand, users must submit their inputs to remote servers, which may expose sensitive data during inference. On the other hand, deploying the model on the user side would reveal proprietary model parameters, which are often obtained through substantial investments in data collection and training. Therefore, enabling secure Transformer inference that simultaneously protects user inputs and model parameters has become an important problem.

Secure multi-party computation (SMPC) provides a promising approach to address these concerns. However, secure generative LLM inference introduces an additional efficiency challenge. Inference typically consists of two phases: prefill and decoding, (see Appendix A.1). The prefill phase processes the full prompt in a single forward pass, whereas the decoding phase generates tokens autoregressively, requiring one forward pass per output token. Since each forward pass in SMPC involves multiple rounds of interaction, the total latency grows with the number of generated tokens. Recent studies (Li et al., 2025) further demonstrate that communication rounds dominate the latency of secure autoregressive decoding.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Party	Scheme	Mul/ARS	Max.	Nonlin.
2	BOLT	26	77	503
2+1	MPCFormer	**13**	99	115
2+1	ARIANN	**13**	3	N/A
2+1	Pika	**13**	N/A	10
2+1	LLAMA	31	**9**	**15**
2+1	Grotto	N/A	N/A	**15**
2+1	SIGMA	49	18	36
3	SPU	49	96	170
3	Mosformer	31	63	84

Mul: secure multiplication.
ARS: secure arithmetic right shift.
Max.: secure maximum computation.
Nonlin.: secure element-wise nonlinear computation.
N/A: not applicable.

Table 1: Rounds per GPT-2-base decoder layer at token 512.

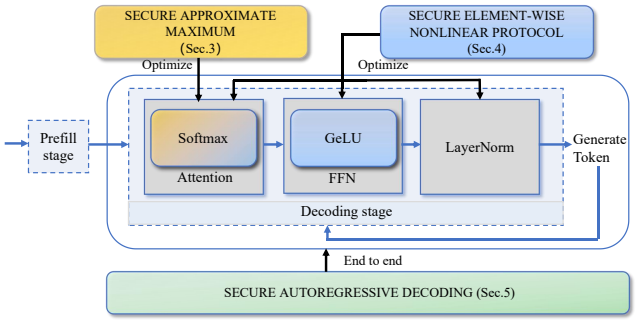


Figure 1: Overview of RESAD.

However, existing representative SMPC-based secure Transformer inference schemes, including BOLT (Pang et al., 2024), MPCFormer (Li et al., 2022), SIGMA (Gupta et al., 2024), and Mosformer (Cheng et al., 2025), are not explicitly designed for round-efficient secure autoregressive decoding. In fact, these schemes mainly focus on classification tasks or the prefill phase of generative tasks, where only a single forward pass is required. As a result, the primary goal of these schemes is to reduce communication overhead, and merely considering the number of communication rounds.

When adopting these schemes to autoregressive decoding, the interaction overhead is repeatedly incurred at each decoding step and accumulates during secure autoregressive decoding, making communication rounds a critical bottleneck. Table 1 reports the communication-round breakdown of a secure GPT2-base decoder layer for generating the 512th token. ARIANN (Ryffel et al., 2022) computes the maximum in three rounds, but its communication cost scales quadratically with the input length. Pika (Wagh, 2022) achieves the lowest round complexity, but its computational cost is too high for nonlinear functions in Transformer inference. Therefore, secure maximum and element-wise nonlinear computation account for a substantial fraction of the total communication rounds. This motivates the following question:

How to design a round-efficient secure autoregressive decoding scheme for transformer inference?

This paper proposes RESAD, a Round-Efficient Secure Autoregressive Decoding Scheme for Transformer Inference, focused on optimizing the round efficiency for secure maximum and element-wise nonlinear computation, where one secure forward pass must be executed for each generated token. While the prefill stage is included for completeness, RESAD mainly optimizes the decoding stage, since its latency accumulates across generation steps and is especially sensitive to communication rounds. Figure 1 illustrates the overall design. Our contributions are summarized as follows:

Constant-round secure approximate maximum protocol for softmax with linear communication complexity. We first propose a secure one-shot bucket-based approximate maximum protocol that replaces exact maximum computation with a round-friendly approximation, further improve its compution efficiency by iterative bucket refinement approximate maximum (Section 3). By computing an approximate maximum rather than the exact maximum, our protocol reduces the round complexity of secure maximum computation from $O(\log m)$ to $O(1)$ for an input of length m, while avoiding the quadratic communication overhead of prior constant-round alternatives (Ryffel et al., 2022). When the input length is $m = 512$, this reduces the communication rounds from 9 to 6 compared with LLAMA (Gupta et al., 2022), while reducing communication by about $300\times$ compared with ARIANN (Ryffel et al., 2022).

Low-round secure element-wise nonlinear protocols based on a new ILUT-based approximation construction. We first propose an interval and table-entry construction, and then combine it with different secure MIC/ILUT protocols to obtain efficient secure protocols for nonlinear functions (Section 4). The resulting protocols reduce the number of AES calls from $O(2^n)$ to approximately $O(2^p(n-p))$ for input in $\mathbb{Z}_{2^n}$, where $p < n$, thus reduce the communication rounds of secure element-wise nonlinear computation from 15 to 7 compared with Grotto (Storrier et al., 2023).

Round-Efficient Secure Autoregressive Decoding Scheme RESAD. We build the RESAD scheme and design secure computation protocols for key Transformer components, including Softmax, GeLU, and LayerNorm (Section 5). Experimental results on Transformer models show that, compared to the state-of-the-art (SOTA), RESAD reduces the communication rounds by 56.4%-63.2%, online communication by 28.9%-51.4%, end to-end runtime by 24.4%-43.3% under wide-area network (WAN) and by 14.5%-16.4% when input 1024th token under local-area network (LAN). Experiments on TinyLlama-1.1B demonstrate the applicability of RESAD to LLaMA-style models.

Related Works. Existing literature on secure Transformer inference can be broadly categorized into several cryptographic paradigms with different trade-offs in computation and communication.

Secret Sharing(SS)-based schemes perform computations over secret shares and optimize nonlinear functions. Built upon CRYPTEN (Knott et al., 2021), prior works explore different design trade-offs. MPCViT (Zeng et al., 2023) studys attention variants to balance efficiency and accuracy, while Ditto (Wu et al., 2024) introduces MPC-friendly quantization. SHAFT (Kei & Chow, 2025) improves efficiency via constant-round Softmax and Fourier-based GeLU approximations.

Function Secret Sharing(FSS)-based schemes have gained increasing attention due to their low online latency via the offline-online paradigm. Pika (Wagh, 2022), Llama (Gupta et al., 2022), Grotto (Storrier et al., 2023), ARIANN (Ryffel et al., 2022), and Orca (Jawalkar et al., 2024) develop FSS-based techniques for secure nonlinear-function evaluation or neural network inference. SIGMA (Gupta et al., 2024) provides the first FSS-based secure Transformer inference scheme with GPU-optimized protocols for Softmax and LayerNorm. Mosformer (Cheng et al., 2025) further extends this line with maliciously secure protocols for comparison and lookup operations.

2 PRELIMINARIES

Notation. Throughout this paper, matrices are denoted by uppercase letters (e.g., $\boldsymbol{W}$), and vectors or individual elements are represented by lowercase letters (e.g., x). Define the bit-widths for the total representation, integer part, and fractional part as n, i, and f, respectively. The indicator function $\mathbf{1}\{x \in S\}$ outputs 1 if and only if $x \in S$, and 0 otherwise. Use Π to denote a secure protocol that outputs a shared value $[x]$, $\hat{\Pi}$ to denote its masked-output variant that outputs a masked value $\hat{x}$, and $\hat{\Pi}^{\mathrm{S2M}}$ to denote the protocol that converts $[x]$ into $\hat{x}$.

Autoregressive Transformer inference. RESAD targets the main nonlinear operators in decoder Transformers: Softmax, GeLU, and LayerNorm (Vaswani et al., 2017). We use the numerically stable form $\mathrm{Softmax}(\boldsymbol{x})_j = \frac{\exp(x_j - \max_k x_k)}{\sum_k \exp(x_k - \max_\ell x_\ell)}$, which makes maximum computation part of every secure attention evaluation. In autoregressive decoding, the current query attends to cached keys and values from the prefix.

Secure-inference setting. The user holds a private input $\boldsymbol{X}$, the model owner holds private parameters $\mathcal{M}$, and two servers P_0, P_1 evaluate the model on additive shares. A trusted third party P_2 supplies input-independent correlated randomness during preprocessing but receives no private input. The model owner distributes shares of $\mathcal{M}$ once, while the user distributes shares of each input and reconstructs the output from the two returned shares. Following prior work (Gupta et al., 2024; Kei & Chow, 2025), we consider a static semi-honest adversary, assume that P_0 and P_1 do not collude, and use secure communication channels. Thus, each server should learn nothing about $\boldsymbol{X}$, $\mathcal{M}$, or the reconstructed output beyond its prescribed view. The security analysis of the secure inference scheme is conducted using the composition theorem (Canetti, 2000), i.e., if each sub-protocol securely realizes its corresponding ideal functionality, then any protocol obtained by sequentially or concurrently composing these sub-protocols also securely realizes the corresponding composed functionality.

FSS-based preprocessed 2PC. A two-party function secret sharing scheme provides algorithms $(\mathrm{Gen}, \mathrm{Eval})$ that split a function g into keys (k_0, k_1) such that, for every input x, $\mathrm{Eval}(0, k_0, x) + \mathrm{Eval}(1, k_1, x) = g(x)$, while either key alone hides the function (Boyle et al., 2015; 2016). We use the resulting offline–online 2PC paradigm (Boyle et al., 2019): before inputs are known, P_2 samples wire masks and distributes FSS keys for the corresponding offset functions; online, P_0 and P_1 exchange only masked wire values and locally evaluate their keys to obtain output shares.

3 CONSTANT-ROUND SECURE APPROXIMATE MAXIMUM

This section presents two approximate maximum methods and their corresponding secure protocols. We first replace exact maximum computation with a one-shot bucket-based approximation, and then refine this design for wide candidate ranges where fine-grained one-shot partitioning is impractical.

3.1 ONE-SHOT BUCKET-BASED APPROXIMATE MAXIMUM

In secure Softmax, the maximum is used for numerical stabilization rather than as an output of the computation. Subtracting it before exponentiation shifts the inputs into a numerically safe range and prevents overflow. Therefore, an exact maximum is unnecessary as long as the shifting value is sufficiently accurate. Approximation alone, however, does not ensure low interaction: a binary-tree maximum still performs sequential pairwise reductions whose depth grows with the input length. We instead seek an approximation that can be evaluated in parallel over all input elements.

This observation motivates bucket localization. We partition the candidate integer range into ordered, disjoint buckets and identify the rightmost bucket that contains at least one input. The exact maximum lies in this bucket, so either endpoint provides an approximation whose error is controlled by the bucket width. As summarized in Algorithm 1, the method proceeds in three stages. First, all input–bucket membership tests are evaluated in parallel. Second, the bucket occupancies are aggregated and encoded into an integer whose most significant nonzero bit identifies the rightmost non-empty bucket. Third, the selected left or right endpoint is returned as the approximate maximum. This workflow replaces sequential pairwise reduction with parallel containment, aggregation, and selection; hence, for a fixed bucket budget, its interaction depth is independent of the input length while its communication remains linear in that length.

The secure realization combines parallel multi-interval containment (Boyle et al., 2021) with secure comparison and selection. Because the most significant nonzero bit is exactly the largest occupied bucket index, the secure selector preserves the error bound of the plaintext construction. The complete protocol appears in Appendix B.

The one-shot design nevertheless exposes a second bottleneck: its cost grows with the number of buckets needed to cover the candidate range at the target precision. For example, a range of width 2^{16} requires about 2^{16} buckets to guarantee unit error. Thus, although one-shot localization removes the input-length dependence in round complexity, applying a fine-grained partition over the entire range can still be prohibitively expensive. This limitation motivates the iterative design below.

3.2 ITERATIVE BUCKET REFINEMENT APPROXIMATE MAXIMUM

To avoid partitioning the full range at the target granularity, we replace the one-shot search with a sequence of coarse-to-fine localizations. The method uses at most $u_{\max}$ buckets in each invocation and maintains an accumulated estimate z. At each intermediate refinement, it applies ApproxMax1 to the residual $\boldsymbol{x} - z$, selects the left endpoint of the occupied bucket, and adds that endpoint to z. This keeps z as a lower-bound estimate while restricting the next refinement to the selected subrange. The final refinement selects the right endpoint, which bounds the approximation error by Δ.

Residualization makes these refinements composable: subtracting the accumulated lower bound translates the selected subrange to zero, allowing the same bounded-size bucket template to be reused at the next scale. Intermediate left endpoints keep z below the exact maximum, whereas the final right endpoint moves it to the upper edge of the last bucket, yielding $0 \le z - \max_k x_k \le \Delta$.

Algorithm 2 formalizes this process. A direct partition would require $u = \lceil (rb - lb)/\Delta \rceil$ target-resolution buckets. We instead choose the smallest refinement depth d for which $u_r = \lceil \sqrt[d]{u} \rceil \le u_{\max}$, and reduce the bucket width at each refinement until the target resolution is reached. This trades a small number of sequential refinements for a bounded localization cost in every refinement.

The public budget $u_{\max}$ trades per-invocation containment work for refinement depth without reintroducing dependence on the input length or secret values.

Each ApproxMax1 invocation requires three rounds, so the complete method costs $3d$ rounds. The value of d depends on $[lb, rb)$, Δ, and $u_{\max}$, but not on the input length. Appendix B gives the complete secure realization.

Algorithm 1 ApproxMax1

Require: vector $\boldsymbol{x} = (x_0, \ldots, x_{m-1})$;
interval endpoints $\boldsymbol{le}, \boldsymbol{re}$;
endpoint flag isLeft.
Ensure: Approximate maximum z

$H_{k,j} \leftarrow \mathbf{1}\{x_k \in [le_j, re_j]\}, \forall k, j$
$t_j \leftarrow \sum_{k=0}^{m-1} H_{k,j}, \forall j$
$tt_j \leftarrow \mathbf{1}\{t_j > 0\}, \forall j$
$y \leftarrow \sum_{j=0}^{u-1} tt_j \cdot 2^j$
$e_j \leftarrow \mathbf{1}\{2^j \le y < 2^{j+1}\}, \forall j$
if isLeft **then**
  $z \leftarrow \sum_{j=0}^{u-1} e_j \cdot le_j$
else
  $z \leftarrow \sum_{j=0}^{u-1} e_j \cdot re_j$
end if
return z

Algorithm 2 ApproxMax

Require: vector $\boldsymbol{x} = (x_0, \ldots, x_{m-1})$; range $[lb, rb)$;
error Δ; interval budget $u_{\max}$.
Ensure: Approximate maximum z

$u \leftarrow \lceil (rb - lb)/\Delta \rceil$
Choose the smallest d such that $\lceil \sqrt[d]{u} \rceil \le u_{\max}$
$u_r \leftarrow \lceil \sqrt[d]{u} \rceil$; $z \leftarrow lb$
for $j = 0$ to $d - 1$ **do**
  $\Delta_j \leftarrow \Delta \cdot u_r^{d-1-j}$
  $\boldsymbol{le}^{(j)} \leftarrow \{0, \Delta_j, \ldots, (u_r - 1)\Delta_j\}$
  $\boldsymbol{re}^{(j)} \leftarrow \{\Delta_j - 1, \ldots, u_r\Delta_j - 1\}$
  if $j < d - 1$ **then**
    $L_j \leftarrow \mathrm{ApproxMax1}(\boldsymbol{x} - z, \boldsymbol{le}^{(j)}, \boldsymbol{re}^{(j)}, 1)$
    $z \leftarrow z + L_j$
  else
    $R_j \leftarrow \mathrm{ApproxMax1}(\boldsymbol{x} - z, \boldsymbol{le}^{(j)}, \boldsymbol{re}^{(j)}, 0)$
    $z \leftarrow z + R_j$
  end if
end for
return z

4 Low-Round Secure Element-Wise Nonlinear Protocols

4.1 Interval and Table-Entry Construction

Element-wise nonlinear functions such as exponential, reciprocal, reciprocal square root, and GeLU are invoked repeatedly across Transformer layers and decoding steps. Although existing secure lookup protocols provide low-round evaluation (Wagh, 2022), their cryptographic cost grows with the number of table entries; directly tabulating an n-bit input domain requires $O(2^n)$ entries. The main challenge is therefore not only to reduce the lookup rounds, but also to reduce the table before applying the secure protocol.

Our key observation is that practical neural inference tolerates limited significant-bit precision, so inputs that differ only in lower insignificant bits need not occupy distinct table entries. We therefore construct an interval lookup table (ILUT) by grouping inputs with the same retained top-p significant-bit pattern and assigning them a shared entry. The resulting representation allocates finer intervals near zero and progressively wider intervals at larger magnitudes, matching the nonuniform resolution induced by significant-bit precision. The parameter p directly controls the accuracy–cost tradeoff; for example, $p = 8$ retains the eight effective significand bits of BF16 arithmetic (Kalamkar et al., 2019).

The significant-bit rule also determines the intervals without exhaustively enumerating the domain. For a positive value whose most significant nonzero bit is at level $j \ge p$, a retained top-p pattern induces an interval of width 2^{j-p}; values with fewer than p significant bits remain singletons. We mirror these intervals for negative inputs. For each resulting interval, we evaluate the target function at its left endpoint, quantize the output, and merge adjacent intervals with identical quantized values. Algorithms 3–5 formalize these three steps: interval generation, table construction and merging, and their composition into an ILUT.

For each of the $n - p$ bit-length levels, this construction produces $O(2^p)$ intervals, while singleton values add only $O(2^p)$ entries. Hence, the number of entries decreases from $O(2^n)$ to approximately $O(2^p(n - p))$, and merging can reduce it further. When p is small, the table therefore depends exponentially on the retained precision rather than the full input bitwidth, substantially reducing the AES calls required by FSS-based lookup.

4.2 Secure Element-Wise Nonlinear Function

Given an input format $(n_{\text{in}}, i_{\text{in}}, f_{\text{in}})$, an output format $(n_{\text{out}}, i_{\text{out}}, f_{\text{out}})$, and a valid range $[lb, rb]$, we first generate the ILUT offline and then securely select the entry corresponding to the secret input. A single lookup realization is not optimal for every Transformer operator because the balance between interaction and local cryptographic work changes with tensor size.

Algorithm 3 GenInterval

Require: bitwidth n_{in}; precision p; range $[lb, rb]$.
Ensure: interval endpoints $(\boldsymbol{le}, \boldsymbol{re})$

Initialize $\boldsymbol{le}, \boldsymbol{re}$ as empty
for $j \leftarrow n_{\text{in}} - 1$ **down to** p **do**
  for $k \leftarrow 2^{p-1}$ **to** $2^p - 1$ **do**
    $L \leftarrow k \cdot 2^{j-p}$; $R \leftarrow L + 2^{j-p} - 1$
    if $[L, R] \subseteq [lb, rb]$ **then**
      append (L, R) to $(\boldsymbol{le}, \boldsymbol{re})$
    end if
    if $[-R, -L] \subseteq [lb, rb]$ **then**
      append $(-R, -L)$ to $(\boldsymbol{le}, \boldsymbol{re})$
    end if
  end for
end for
for $v \leftarrow 1$ **to** $2^{p-1} - 1$ **do**
  if $v \in [lb, rb]$ **then**
    append (v, v) to $(\boldsymbol{le}, \boldsymbol{re})$
  end if
  if $-v \in [lb, rb]$ **then**
    append $(-v, -v)$ to $(\boldsymbol{le}, \boldsymbol{re})$
  end if
end for
if $0 \in [lb, rb]$ **then**
  append $(0, 0)$ to $(\boldsymbol{le}, \boldsymbol{re})$
end if
Sort intervals by left endpoint
return $(\boldsymbol{le}, \boldsymbol{re})$

Algorithm 4 GenTable

Require: intervals $(\boldsymbol{le}, \boldsymbol{re})$; bitwidth n_{out}; fractional bits $f_{\text{in}}, f_{\text{out}}$; function $\mathcal{F}$.
Ensure: merged endpoints $(\boldsymbol{le}', \boldsymbol{re}')$, table $\boldsymbol{T}$

Initialize $\boldsymbol{le}', \boldsymbol{re}', \boldsymbol{T}$ as empty
$T_{\min}, T_{\max} \leftarrow -2^{n_{\text{out}}-1}, 2^{n_{\text{out}}-1} - 1$
for each interval j **do**
  $x \leftarrow le_j / 2^{f_{\text{in}}}$
  $t \leftarrow \text{round}(\mathcal{F}(x) \cdot 2^{f_{\text{out}}})$
  if $t \notin [T_{\min}, T_{\max}]$ **then**
    continue
  end if
  if $\boldsymbol{T} \neq \emptyset$ **and** $t = T_{\text{last}}$ **then**
    $re'_{\text{last}} \leftarrow re_j$
  else
    append le_j, re_j, t to $\boldsymbol{le}', \boldsymbol{re}', \boldsymbol{T}$
  end if
end for
return $(\boldsymbol{le}', \boldsymbol{re}'), \boldsymbol{T}$

Algorithm 5 GenILUT

Require: bitwidths $n_{\text{in}}, n_{\text{out}}$; fractional bits $f_{\text{in}}, f_{\text{out}}$; precision p; function $\mathcal{F}$; range $[lb, rb]$.
Ensure: endpoints $(\boldsymbol{le}, \boldsymbol{re})$, table $\boldsymbol{T}$

$(\boldsymbol{le}, \boldsymbol{re}) \leftarrow \text{GenInterval}(n_{\text{in}}, p, [lb, rb])$
$(\boldsymbol{le}, \boldsymbol{re}), \boldsymbol{T} \leftarrow$
  $\text{GenTable}((\boldsymbol{le}, \boldsymbol{re}), n_{\text{out}}, f_{\text{in}}, f_{\text{out}}, \mathcal{F})$
return $(\boldsymbol{le}, \boldsymbol{re}), \boldsymbol{T}$

Both protocols consume the same offline-generated ILUT and differ only in the secure lookup backend used online. For low-dimensional inputs such as reciprocal and reciprocal square root, Π^{ILUT1} directly combines the ILUT with Boyle et al.'s one-round MIC (Boyle et al., 2021), prioritizing minimum interaction. Exponential and GeLU operate on much larger tensors, for which the local computation of secure lookup becomes more important. Protocol Π^{ILUT2} therefore uses Grotto's two-round construction (Storrier et al., 2023): it introduces one additional interaction round but reduces the lookup computation. This division lets each operator balance network interaction against local cryptographic work. Appendix C gives both complete protocols.

5 Secure Autoregressive Decoding

This section composes the preceding techniques into secure Softmax, GeLU, LayerNorm, and autoregressive decoding. The remaining building blocks and the complete protocols appear in Appendices D and E, respectively.

Secure Softmax Protocol. Given an input vector $\hat{\boldsymbol{x}}$, the secure Softmax computes $\text{Softmax}(\boldsymbol{x})$ under fixed-point secret sharing. Our design follows the standard maximum-subtraction formulation, but adapts it to a low-round secure setting.

Before maximum computation, the input is locally right-shifted by $f - 8$ bits to retain eight effective fractional bits. This scaling matches the target approximation error $\Delta = 0.5$, which corresponds to 2^7 in the integer domain. After obtaining $\hat{\tilde{x}}$, the protocol evaluates $\exp(\hat{\boldsymbol{x}} - \hat{\tilde{x}})$ using Π^{ILUT2} and the reciprocal of the exponential sum using Π^{ILUT1}.

For the exponential, we use an ILUT with $p = 8$ over $[-2^{f+4}, 0]$, corresponding to the real interval $[-16, 0]$. The reciprocal range is $[2^{f-1}, 2^f \text{len}(\boldsymbol{x})]$, because the exponential sum lies between 0.5 and $\text{len}(\boldsymbol{x})$. The protocol then multiplies by the reciprocal and locally shifts the result back to the target fixed-point format.

Secure GeLU Protocol. Our secure GeLU protocol follows the decomposition used in prior work such as SIGMA: $\mathrm{GeLU}(x) = \mathrm{ReLU}(x) + \delta(x),\ \delta(x) = \mathrm{GeLU}(x) - \mathrm{ReLU}(x)$. The correction term is significant only near the origin, so we restrict its effective domain to $[-2^{f+2}, 2^{f+2})$, corresponding to $[-4, 4)$. The protocol evaluates $\delta(\cdot)$ with the interval lookup and obtains the ReLU term from a secure comparison with zero; adding the two terms yields the GeLU output.

Secure LayerNorm Protocol. Given $\hat{\boldsymbol{x}}$ and affine parameters $\hat{\boldsymbol{\gamma}}$ and $[\boldsymbol{\beta}]$, the protocol securely computes the mean and variance using summation and fixed-point averaging. It evaluates the reciprocal square root of the variance with Π^{ILUT1}, normalizes $\boldsymbol{x} - \mu$, and applies the learnable affine transformation.

Autoregressive generation consists of an initial prefill followed by iterative decoding. Each secure step processes only the newly generated token, appends its per-layer keys and values to the KV cache without additional interaction, and attends the current query to all cached keys. The resulting logits are revealed to the user, who selects the next token locally and returns it in masked form for the next step. This avoids recomputing the prefix while the attention score vector grows linearly with the context length.

6 Evaluation

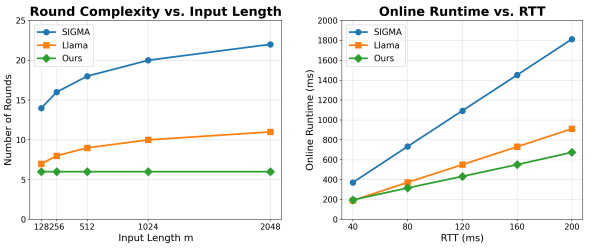
Experimental Setup. We use one Alibaba Cloud ECS instance (`ecs.gn8is.4xlarge`) with 16 Intel Xeon Gold 6462C–based vCPUs, 128 GiB RAM, and a 48 GB NVIDIA L20 GPU under Ubuntu 22.04.5. Linux Traffic Control emulates LAN (1 Gbps, 0.3 ms RTT) and WAN (100 Mbps, 80 ms RTT), and fixed-point values use $(n, i, f) = (64, 16, 12)$. We evaluate pretrained Hugging Face models (Wolf et al., 2020): GPT-2 Small, GPT-2 Medium, and TinyLlama-1.1B, plus Cerebras-GPT-256M for accuracy. Microbenchmarks use GPT-2 Small shapes at token 512. For end-to-end decoding, we measure steps that take the 128th, 512th, and 1024th tokens as input and produce the 129th, 513th, and 1025th tokens, respectively. Accuracy is evaluated on LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2020), and HellaSwag (Zellers et al., 2019).

6.1 Microbenchmarks

Secure Maximum. We compare with SIGMA (Gupta et al., 2024), LLAMA (Gupta et al., 2022), and ARIANN (Ryffel et al., 2022) at shape $(12, 512)$. We also vary the input length and RTT to evaluate round and runtime scaling. Under the parameter setting in Section 5, our protocol invokes $\hat{\Pi}^{\mathrm{ApproxMax1}}$ twice and therefore requires 6 rounds.

Table 2: Communication and rounds of secure maximum

Maximum $(12, 512)$			
Scheme	Comm.		Rds.
	Online	Total	
SIGMA	38.93 KB	3.85 MB	18
LLAMA	37.43 KB	7.33 MB	9
ARIANN	7.52 MB	3.03 GB	3
Ours	**6.02 KB**	11.15 MB	6



(a) Communication rounds under different input lengths.

(b) Online runtime under different RTT settings.

Figure 2: Round and runtime scaling of secure maximum.

Table 2 reports 6 rounds and 6.02 KB of online communication, with over two orders of magnitude less total communication and 99.9% less online communication than constant-round ARIANN. Figure 2(a) confirms that the round complexity remains constant at 6 as the input grows. Figure 2(b) further shows approximately linear runtime growth with RTT; our online runtime is 47.1%–62.8% and 15.1%–26.0% lower than SIGMA and LLAMA, respectively.

Secure Nonlinear Functions. We compare secure exponential (Exp), reciprocal, and reciprocal square root (Rsqrt) with SIGMA (Gupta et al., 2024) and Grotto (Storrier et al., 2023); because Grotto has no public implementation, we simulate its reported 12-bit configuration with ten degree-2 spline segments. This comparison tests whether the compact ILUT construction reduces both interaction and lookup cost across operators with different input dimensions.

Table 3: Communication, rounds, and WAN/LAN runtime of secure nonlinear functions

Operator	Scheme	Rds.	Comm.		WAN Time (ms)		LAN Time (ms)	
			Online	Total	Online	Total	Online	Total
Exp $(12, 512)$	SIGMA	11	808.5 KB	8.55 MB	483.48	1579.94	14.07	86.85
	Grotto	3	576 KB	5.81 MB	150.15	1005.36	**8.36**	58.33
	Ours	**2**	**288 KB**	**4.73 MB**	**121.20**	**892.07**	25.49	**56.33**
Reciprocal (12)	SIGMA	4	628 B	**5.54 KB**	263.34	264.45	111.72	112.62
	Grotto	3	1152 B	11.63 KB	124.17	126.55	4.84	6.43
	Ours	**1**	**192 B**	13.78 KB	**41.65**	**43.29**	**2.61**	**3.36**
Rsqrt (1)	SIGMA	7	84 B	1.56 KB	288.42	289.87	13.42	14.79
	Grotto	3	96 B	**0.97 KB**	122.93	124.49	4.68	6.20
	Ours	**1**	**16 B**	1.38 KB	**41.76**	**42.79**	**2.60**	**3.48**

As Table 3 shows, Exp requires 2 rounds, while Reciprocal and Rsqrt require only 1 round; all three protocols use less online communication than both baselines. Compared with SIGMA, they reduce WAN online and total runtime by 74.9%–85.5% and 43.5%–85.2%, respectively. Relative to Grotto, the corresponding reductions are 19.3%–66.5% and 11.3%–65.8%. In LAN, our protocols also achieve lower total runtime for Reciprocal and Rsqrt; complete results appear in Appendix F.

6.2 Comparison of Secure Operators

We further compare complete GPT-2 Small Softmax, GeLU, and LayerNorm operators with SIGMA (Gupta et al., 2024) to determine whether the component-level improvements persist after composition.

Table 4: Communication, rounds, and WAN/LAN runtime of secure Transformer operators

Operator	Scheme	Rds.	Comm.		WAN Time (ms)		LAN Time (ms)	
			Online	Total	Online	Total	Online	Total
Softmax $(12, 512)$	SIGMA	36	1.01 MB	**13.44 MB**	1582.93	3156.80	148.75	269.66
	Ours	**10**	**0.38 MB**	15.50 MB	**506.48**	**2313.15**	**112.12**	**262.66**
GeLU $(1, 3072)$	SIGMA	7	216.75 KB	**2.32 MB**	336.57	842.49	**10.90**	**31.88**
	Ours	**2**	**144.75 KB**	2.41 MB	**121.74**	**644.24**	36.17	58.39
LayerNorm $(1, 768)$	SIGMA	10	24.10 KB	44.57 KB	411.48	456.36	15.26	31.14
	Ours	**4**	**24.03 KB**	**43.95 KB**	**165.42**	**208.84**	**4.61**	**7.01**

Table 4 shows that our protocols reduce online communication by 62.4%, 33.2%, and 0.3%, and rounds by 72.2%, 71.4%, and 60.0% for Softmax, GeLU, and LayerNorm, respectively. Under WAN, online runtime decreases by 59.8%–68.0% and total runtime by 23.5%–54.2%. These results show that the lower interaction complexity persists after operator composition; full LAN results appear in Appendix F.

6.3 Secure Autoregressive Decoding

We evaluate end-to-end performance and accuracy on GPT-2 Small, GPT-2 Medium, and TinyLlama-1.1B. Following Cachemir (Yu et al., 2026), we replace SIGMA's nonlinear components with ours and compare one-step decoding with SIGMA (Gupta et al., 2024) and SHAFT (Kei & Chow, 2025) using their public implementations. We recompute SHAFT's runtime with bandwidth units consistent with its stated settings; see Appendix H.

End-to-End Performance. We measure rounds, communication, and runtime at tokens 128, 512, and 1024 to capture scaling with context length.

Table 5: End-to-end cost of secure autoregressive decoding

Model	Scheme	Rds.	Comm. (MB)		WAN (s)		LAN (s)	
			Online	Total	Online	Total	Online	Total
GPT-2 Small 128th	SHAFT	1445	/	2580.48	/	405.41	/	26.22
	SIGMA	793	9.42	**73.95**	33.07	39.48	**1.52**	**2.33**
	Ours	**319**	**6.70**	88.93	**14.53**	**22.39**	2.09	3.05
GPT-2 Small 512th	SHAFT	1445	/	2792.45	/	417.49	/	27.91
	SIGMA	841	19.41	**197.01**	36.24	52.21	2.69	4.52
	Ours	**319**	**10.93**	222.60	**14.79**	**33.68**	**2.27**	**4.39**
GPT-2 Small 1024th	SHAFT	1445	/	3075.07	/	437.45	/	30.18
	SIGMA	865	32.74	**361.09**	39.26	68.48	4.23	7.42
	Ours	**319**	**16.56**	399.48	**15.22**	**49.51**	**2.56**	**6.20**
GPT-2 Medium 128th	SHAFT	2861	/	6462.46	/	915.83	/	64.54
	SIGMA	1573	22.51	**193.89**	65.75	81.67	**3.15**	**5.13**
	Ours	**631**	**15.27**	233.85	**28.81**	**48.37**	4.17	6.57
GPT-2 Medium 512th	SHAFT	2861	/	7026.69	/	947.12	/	69.23
	SIGMA	1669	49.16	**522.06**	72.75	114.02	5.92	10.62
	Ours	**631**	**26.55**	590.31	**29.51**	**78.56**	**4.64**	**10.14**
GPT-2 Medium 1024th	SHAFT	2861	/	7780.35	/	995.60	/	75.43
	SIGMA	1717	84.69	**959.60**	79.68	157.70	9.52	17.83
	Ours	**631**	**41.57**	1061.97	**30.67**	**119.16**	**5.66**	**15.25**
TinyLlama-1.1B 128th	SHAFT	–	–	–	–	–	–	–
	SIGMA	1531	37.33	**322.18**	65.46	91.14	**4.37**	**7.58**
	Ours	**667**	**25.29**	385.14	**31.20**	**63.89**	5.30	9.18
TinyLlama-1.1B 512th	SHAFT	–	–	–	–	–	–	–
	SIGMA	1619	86.19	**923.83**	74.25	150.26	8.32	16.48
	Ours	**667**	**45.97**	1038.64	**32.58**	**119.37**	**6.45**	**15.97**
TinyLlama-1.1B 1024th	SHAFT	–	–	–	–	–	–	–
	SIGMA	1663	151.33	**1726.00**	83.62	224.49	13.58	28.29
	Ours	**667**	**73.50**	1903.37	**34.32**	**192.09**	**7.76**	**24.74**

Note: For SHAFT, "/" denotes an unavailable online-phase communication breakdown, while "–" denotes GPU memory overflow.

As shown in Table 5, RESAD reduces rounds by 56.4%–63.2% and online communication by 28.9%–51.4% relative to SIGMA. On the GPT-2 models, it reduces rounds by 77.9% and total communication by 86.4%–96.6% relative to SHAFT. These results confirm that RESAD lowers the online interaction required by secure decoding across both GPT-2 and LLaMA-style architectures.

Under WAN, RESAD reduces total runtime by 24.4%–43.3% on the GPT-2 models and by 14.4%–29.9% on TinyLlama-1.1B; relative to SHAFT, the reduction on GPT-2 is 88.03%–94.72%. In LAN, local computation is more visible at the 128th token, but the length-dependent cost of SIGMA grows with context length. RESAD therefore becomes faster than SIGMA at the 512th and 1024th tokens for all three models. Detailed online runtimes appear in Appendix F.

Accuracy Evaluation. We compare secure and plaintext inference to measure the effect of our approximations on model quality. As shown in Table 6, secure inference maintains accuracy comparable to plaintext inference across all datasets and models, demonstrating that our optimizations do not introduce noticeable degradation in model quality.

Table 6: Accuracy of plaintext and secure inference

Model	LAMBADA		PIQA		HellaSwag	
	Plain	Ours	Plain	Ours	Plain	Ours
GPT-2 Small	32.56%	32.19%	62.89%	63.11%	28.92%	29.01%
GPT-2 Medium	42.98%	42.75%	67.63%	67.19%	33.31%	33.32%
Cerebras-GPT-256M	29.36%	30.39%	61.37%	61.92%	27.44%	27.51%
TinyLlama-1.1B	61.01%	60.24%	74.54%	74.16%	46.59%	46.12%

7 Conclusion

We presented RESAD, a round-efficient secure autoregressive decoding scheme combining a constant-round approximate-maximum protocol with low-round ILUT protocols. RESAD reduces rounds by 56.4%–63.2% and WAN runtime by 24.4%–43.3% relative to SIGMA while maintaining accuracy comparable to plaintext inference. The experimental results show that RESAD is overwhelming other secure transformer inference schemes and is suitable for long-content generation.

AI use statement

During the preparation of this manuscript, the authors used ChatGPT to improve the grammar, wording, and readability of author-written drafts and help write scripts used for experimental evaluation.

The authors reviewed the AI-assisted evaluation scripts for correctness before they were used. The authors reviewed and revised all other AI-assisted changes and take full responsibility for the final content of this submission.

ETHICS STATEMENT

We adhere to ethical principles by clearly identifying stakeholders, including users, developers, the academic community, and the broader public, ensuring their rights and interests are respected; upholding Beneficence by striving to enhance societal well-being through technological innovation while mitigating potential harms; and practicing Respect for Persons by honoring individual autonomy and privacy, maintaining transparency and fairness in the research process, and safeguarding participants' right to informed consent. We have taken the following measures to ensure the ethical integrity of our research:

- Awareness of Ethical Guidelines: The authors have read and understood the ethics considerations outlined in the conference call for papers, the detailed submission instructions, and the guidelines for ethics documentation.
- Ethical Considerations and Conduct: The research team carefully considered the ethical implications of this study. The authors believe that the research was conducted ethically, and the team's next-step plans (e.g., following publication) are also aligned with ethical standards.

REPRODUCIBILITY STATEMENT

The source code is available at `https://anonymous.4open.science/r/resad-7C54`. Protocol constructions, experimental settings, and additional evaluation results are documented in the main paper and appendix.

REFERENCES

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*, pp. 337–367. Springer, 2015.

Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 1292–1303, 2016.

Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography Conference*, pp. 341–371. Springer, 2019.

Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 871–900. Springer, 2021.

Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

Ke Cheng, Yuheng Xia, Anxiao Song, Jiaxuan Fu, Wenjie Qu, Yulong Shen, and Jiaheng Zhang. Mosformer: Maliciously secure three-party inference framework for large transformers. *Cryptology ePrint Archive*, 2025.

Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. Llama: A low latency math library for secure inference. *Cryptology ePrint Archive*, 2022.

Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 4:61ś79, 2024.

Neha Jawalkar, Kanav Gupta, Arkaprava Basu, Nishanth Chandran, Divya Gupta, and Rahul Sharma. Orca: Fss-based secure training and inference with gpus. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 597–616. IEEE, 2024.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

Andes YL Kei and Sherman SM Chow. Shaft: Secure, handy, accurate, and fast transformer inference. In *Network and Distributed System Security Symposium, NDSS*, volume 2025, 2025.

Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems*, 34:4961–4973, 2021.

Dacheng Li, Rulin Shao, Hongyi Wang, Han Guo, Eric P Xing, and Hao Zhang. Mpcformer: fast, performant and private transformer inference with mpc. *arXiv preprint arXiv:2211.01452*, 2022.

Zhengyi Li, Yue Guan, Kang Yang, Yu Feng, Ning Liu, Yu Yu, Jingwen Leng, and Minyi Guo. An efficient private gpt never autoregressively decodes. *arXiv preprint arXiv:2505.15252*, 2025.

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *IEEE Symposium on Security and Privacy (SP)*, pp. 4753–4771. IEEE, 2024.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis Bach. Ariann: Low-interaction privacy-preserving deep learning via function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 1:291–316, 2022.

Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. Grotto: Screaming fast (2+1)-pc for $\mathbb{Z}_{2^n}$ via (2,2)-dpfs. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2143–2157, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies*, 2022.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Haoqi Wu, Wenjing Fang, Yancheng Zheng, Junming Ma, Jin Tan, and Lei Wang. Ditto: Quantization-aware secure inference of transformers upon mpc. In *International Conference on Machine Learning*, pp. 53346–53365. PMLR, 2024.

Ye Yu, Yifan Zhou, Yi Chen, Pedro Soto, Wenjie Xiong, and Meng Li. Cachemir: Fully homomorphic encrypted inference of generative large language model with kv cache. *arXiv preprint arXiv:2602.11470*, 2026.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Wenxuan Zeng, Meng Li, Wenjie Xiong, Tong Tong, Wen-jie Lu, Jin Tan, Runsheng Wang, and Ru Huang. Mpcvit: Searching for accurate and efficient mpc-friendly vision transformer with heterogeneous attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5052–5063, 2023.

A Extended Preliminaries

A.1 Transformer Architecture

Transformer Components. Transformer first maps discrete input tokens into continuous representations through an embedding layer, and then processes them using a stack of identical Transformer layers. Each layer consists of a multi-head attention (MHA) module followed by a feed-forward network (FFN), with residual connections and layer normalization (LayerNorm) applied to stabilize training and inference.

Embedding. Given a token sequence $\boldsymbol{s} = (s_0, \ldots, s_{m-1})$, the token embedding layer can be written as

$$\boldsymbol{Y} = \mathrm{Embed}(\boldsymbol{s}) = \boldsymbol{X}\boldsymbol{E} + \boldsymbol{P},$$

where $\boldsymbol{X} \in \mathbb{R}^{m\times|\mathcal{V}|}$ is the one-hot matrix, $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{V}|\times d}$ is the token embedding matrix, and $\boldsymbol{P} \in \mathbb{R}^{m\times d}$ is the positional embedding matrix, whose i-th row corresponds to position i for $i = 0, \ldots, m-1$. The resulting $\boldsymbol{Y} \in \mathbb{R}^{m\times d}$ is used as the input to the first Transformer layer.

Multi-Head Attention. Given $\boldsymbol{X} \in \mathbb{R}^{m\times d}$, MHA first projects it into queries, keys, and values:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V.$$

For autoregressive decoding, a causal mask $\boldsymbol{M} \in \mathbb{R}^{m\times m}$ is applied to prevent attending to future tokens. The attention output is

$$\mathrm{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathrm{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}} + \boldsymbol{M}\right)\boldsymbol{V},$$

where $M_{i,j} = -\infty$ if $j > i$, and 0 otherwise. MHA applies this operation across multiple heads and concatenates the results.

Feed-Forward Network. The FFN applies two linear transformations with a GeLU activation:

$$\mathrm{FFN}(\boldsymbol{x}) = \boldsymbol{W}_2\left(\mathrm{GeLU}\left(\boldsymbol{X}\boldsymbol{W}_1 + \boldsymbol{B}_1\right)\right) + \boldsymbol{B}_2.$$

Layer Normalization. LayerNorm normalizes each input vector across its feature dimension:

$$\mathrm{LayerNorm}(\boldsymbol{x}) = \boldsymbol{\gamma} \odot \frac{\boldsymbol{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \boldsymbol{\beta},$$

where $\mu = \frac{1}{d}\sum_{i=0}^{d-1} x_i$ and $\sigma^2 = \frac{1}{d}\sum_{i=0}^{d-1}(x_i - \mu)^2$ are the mean and variance, $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^d$ are learnable parameters, and ϵ is a small constant for numerical stability.

Generative LLM Inference. Generative LLMs perform inference in an autoregressive manner, where tokens are generated sequentially and each step depends on all previously generated tokens. Given an input prompt $\boldsymbol{x}_{0:m-1}$, the model iteratively produces output tokens $x_m, x_{m+1}, \ldots$.

At each decoding step t, the model computes logits $\boldsymbol{z}_t \in \mathbb{R}^{|\mathcal{V}|}$ over the vocabulary $\mathcal{V}$ via a Transformer forward pass:

$$\boldsymbol{z}_t = \mathcal{M}(\boldsymbol{x}_{0:m+t-1}).$$

The resulting representation is used to produce logits $\boldsymbol{z}_t$, and the next token is selected as $x_{m+t} = \mathrm{Sample}(\boldsymbol{z}_t)$, where $\mathrm{Sample}(\cdot)$ can be implemented as greedy decoding (e.g., $\arg\max$) or stochastic sampling methods.

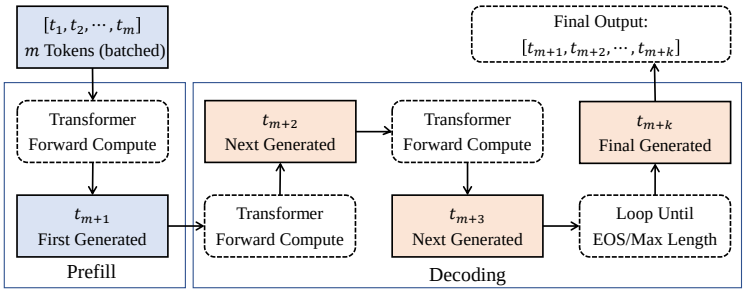


Figure 3: Prefill and autoregressive decoding.

A.2 Cryptographic Primitives

We consider a set of two primary participants $\{P_0, P_1\}$. During the offline phase, a Trusted Third Party (TTP), denoted as P_2, is responsible for generating correlated randomness.

Secret Sharing. Secret sharing is a fundamental primitive in SMPC. Given a value $x \in \mathbb{Z}_{2^n}$, it can be split into two additive shares (x_0, x_1) such that $x = x_0 + x_1 \pmod{2^n}$, where each share individually reveals no information about x. The shares are distributed to two non-colluding parties P_0, P_1.

Function Secret Sharing. Function Secret Sharing (FSS) is a set of functions $(\mathrm{Gen}, \mathrm{Eval})$. The Gen algorithm splits a secret function $g : \mathbb{G}^{\mathrm{in}} \to \mathbb{G}^{\mathrm{out}}$ into g_0 and g_1. For the b-th participant P_b, the Eval algorithm executes the function g_b on an input $x \in \mathbb{G}^{\mathrm{in}}$. The correctness requirement of FSS is that for all $x \in \mathbb{G}^{\mathrm{in}}$, it holds that $g_0(x) + g_1(x) = g(x)$. The security requirement is that g_b conceals g.

Definition 1 *Function Secret Sharing. A 2-party function secret sharing scheme (Boyle et al., 2015; 2016) consists of a set of functions* $(\mathrm{Gen}, \mathrm{Eval})$*, where:*

$\mathrm{Gen}(1^\lambda, \hat{g})$ *is a probabilistic polynomial-time key generation algorithm that takes as input* 1^λ *and a description* $\hat{g} \in \{0,1\}^*$ *of the function* f*, and outputs a key pair* (k_0, k_1)*. Here,* $\hat{g}$ *explicitly includes descriptions of the groups* $\mathbb{G}^{\mathrm{in}}$ *and* $\mathbb{G}^{\mathrm{out}}$ *to which the input and output elements belong.*

$\mathrm{Eval}(b, k_b, x)$ *is a polynomial-time algorithm that takes as input the participant index* $b \in \{0,1\}$*,* k_b*, and* $x \in \mathbb{G}^{\mathrm{in}}$*, and outputs* $g_b(x) \in \mathbb{G}^{\mathrm{out}}$*.*

Secure 2PC with Preprocessing from FSS. Building on FSS, secure two-party computation can be realized in an offline–online paradigm (Boyle et al., 2019).

Offline phase. For each wire w_j in the circuit, a random mask $r_j \in \mathbb{G}$ is sampled. For each gate $g : \mathbb{G}^{\mathrm{in}} \to \mathbb{G}^{\mathrm{out}}$ with input wire w_j and output wire w_k, the dealer constructs an offset function $g_{[r_j, r_h]}(x) = g(x - r_j) + r_h$ and generates FSS keys $(k_0, k_1) \leftarrow \mathrm{Gen}(g_{[r_j, r_h]})$, distributing k_b to party P_b. This preprocessing depends only on the circuit structure and is independent of actual inputs.

Online phase. For an input $[x_j]_b$ owned by P_b, P_b computes and sends $[\hat{x}_j]_b = [x_j]_b + [r_j]_b$, so that both parties obtain the masked input. We denote the FSS-based gate-evaluation protocol by Π, which maps $\hat{x}_j$ to $\hat{x}_h = \hat{\Pi}(\hat{x}_j)$. The parties evaluate gates in topological order. For each gate, P_b locally computes $y_b = \Pi_b(\hat{x}_j) = \mathrm{Eval}(b, k_b, \hat{x}_j)$ as its additive share of $\hat{\Pi}(\hat{x}_h)$.

Threat Model. Following the conventions of prior works (Gupta et al., 2024; Kei & Chow, 2025), we assume a semi-honest adversary model. In this setting, all participants are assumed to strictly adhere to the protocol specifications but may attempt to infer additional information about private inputs or model weights from the observed transcript. We further assume that P_0 and P_1 do not collude. All communication channels between participants are secured to prevent eavesdropping and external tampering.

A.3 SMPC-Based Secure Inference with an Online–Offline Paradigm

We consider a standard two-party outsourced secure inference setting with an additional TTP. The system involves a user holding private input $\boldsymbol{X}$, a model owner holding model parameters $\mathcal{M}$, two non-colluding semi-honest servers P_0 and P_1, and a TTP P_2 that assists in preprocessing.

In the offline phase, P_2 generates correlated randomness independent of the inputs and distributes it to P_0 and P_1. The model owner secret-shares the model parameters as $[\mathcal{M}] = ([\mathcal{M}]_0, [\mathcal{M}]_1)$ and sends them to the two servers.

In the online phase, the user secret-shares the input as $[\boldsymbol{X}] = ([\boldsymbol{X}]_0, [\boldsymbol{X}]_1)$ and sends the shares to P_0 and P_1. The two servers then jointly evaluate the model over secret shares

$$[\boldsymbol{Y}] = \mathcal{M}([\boldsymbol{X}]),$$

by invoking a sequence of secure protocols for linear and nonlinear operations. At the end of the computation, each server obtains a share of the output, denoted by $[\boldsymbol{Y}]_0$ and $[\boldsymbol{Y}]_1$, which are sent back to the user. The user reconstructs the final result as

$$\boldsymbol{Y} = [\boldsymbol{Y}]_0 + [\boldsymbol{Y}]_1 \pmod{2^n}.$$

Throughout the process, neither P_0 nor P_1 learns any information about $\boldsymbol{X}$, $\mathcal{M}$, or $\boldsymbol{Y}$ beyond their own shares, while the user learns only the final output.

B Details of Constant-Round Secure Approximate Maximum

B.1 Design Overview

Constant-round secure approximate maximum. We design a new approximate maximum method and develop a constant-round secure protocol. Since softmax stabilization only requires a shifting value that moves the inputs into a numerically safe range, rather than the exact maximum itself, an approximate maximum can be sufficient for practical inference as long as the induced stabilization error is well controlled. This observation serves as the starting point for addressing the first limitation.

Specifically, we partition the candidate value range into a sequence of ordered intervals, each treated as a bucket. As illustrated in Fig. 4, instead of performing sequential pairwise comparisons as in binary-tree-based constructions, our method localizes all inputs into these buckets in parallel and uses the rightmost non-empty bucket to derive an approximate maximum. For example, the inputs $[2.3]$ and $[2.5]$ are mapped to the bucket $[0, 4)$, while $[6.2]$ and $[4.6]$ are mapped to the bucket $[4, 8)$. Since $[4, 8)$ is the rightmost non-empty bucket, its representative value is used as the approximate maximum.

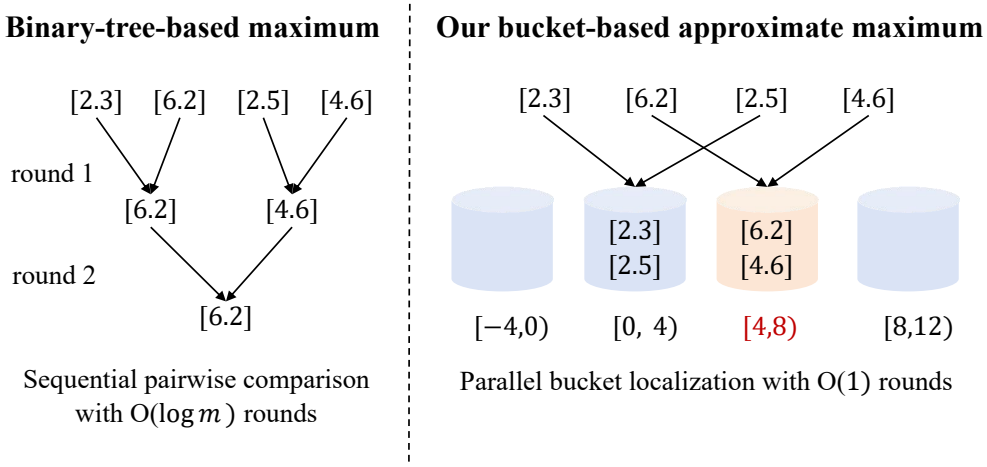


Figure 4: Comparison between binary-tree-based maximum and our bucket-based approximate maximum

At the protocol level, we build on Boyle et al.'s secure multi-interval containment (MIC) protocol Boyle et al. (2021) to securely identify bucket membership and the rightmost non-empty bucket. In this way, sequential reduction over input elements is replaced by parallel bucket localization, reducing the round complexity from $O(\log m)$ to $O(1)$ for an input vector of length m, while preserving linear communication complexity.

B.2 Secure Protocols

Secure Protocol. Fig. 5 presents the secure realization of ApproxMax1. The protocol instantiates the interval-localization procedure in Algorithm 1 using secure interval-containment, comparison, and conversion primitives. Compared with the plaintext algorithm, the main additional issue is to control intermediate bitwidths so as to avoid overflow without introducing unnecessary communication. In particular, since the occupancy count of each interval ranges from 0 to m, it requires $n_t = \lceil \log_2(m+1) \rceil$ bits.

$\Pi^{\mathrm{ApproxMax1}}_{n_{\mathrm{in}}, n_{\mathrm{out}}, \boldsymbol{le}, \boldsymbol{re}, \mathsf{isLeft}}(\hat{\boldsymbol{x}})$

$m \leftarrow \mathrm{len}(\hat{\boldsymbol{x}})$
$u \leftarrow \mathrm{len}(\boldsymbol{le})$
$n_t \leftarrow \lceil \log_2(m+1) \rceil$
$[\boldsymbol{H}] \leftarrow \Pi^{\mathrm{MIC(Boyle)}}_{n_{\mathrm{in}}, n_t, \boldsymbol{le}, \boldsymbol{re}}(\hat{\boldsymbol{x}})$
Compute $[\boldsymbol{t}]$ where $[t_j] = \sum_{k=0}^{m-1} [H_{k,j}],\ \forall j \in \{0, \ldots, u-1\}$
$\hat{\boldsymbol{t}} \leftarrow \hat{\Pi}^{\mathrm{S2M}}_{n_t}([\boldsymbol{t}])$
$[tt_i] \leftarrow \Pi^{\mathrm{Greater}}_{n_t, u}(\hat{t}_i, 0),\ \forall i \in \{0, \ldots, u-1\}$
$\hat{y} \leftarrow \hat{\Pi}^{\mathrm{S2M}}_{u}(\sum_{i=0}^{u-1} [tt_i] \cdot 2^i)$
$[\boldsymbol{e}] \leftarrow \Pi^{\mathrm{MIC(Boyle)}}_{u, n_{\mathrm{out}}, \{2^0, \cdots, 2^{u-1}\}, \{2^1-1, \cdots, 2^u-1\}}(\hat{y})$
if isLeft **then**
    $[z] \leftarrow \sum_{j=0}^{u-1} ([e_j] \cdot le_j)$
else
    $[z] \leftarrow \sum_{j=0}^{u-1} ([e_j] \cdot re_j)$
end if
return $[z]$

Figure 5: Secure one-shot bucket-based approximate maximum protocol

Secure Protocol. Fig. 6 presents the secure realization of the iterative refinement method. The protocol repeatedly invokes the secure one-shot protocol on progressively refined residual ranges, while keeping the number of intervals in each round bounded by $u_{\max}$. The selected interval endpoints are accumulated to obtain the final approximate maximum.

The communication round of our protocol depends on $[lb, rb)$, Δ, and $u_{\max}$. Specifically, each evaluation of $\hat{\Pi}^{\mathrm{ApproxMax1}}$ requires 3 rounds. Based on the above parameters, we can determine the number of required invocations, denoted by d. Therefore, the total communication round complexity of the protocol is $3d$ rounds.

$\Pi^{\mathrm{ApproxMax}}_{n_{\mathrm{in}},n_{\mathrm{out}},lb,rb,\Delta,u_{\max}}(\hat{\boldsymbol{x}})$

$u \leftarrow \lceil (rb - lb)/\Delta \rceil$
$n \leftarrow \max(n_{\mathrm{in}}, n_{\mathrm{out}})$
$d \leftarrow 1$
while $\lceil \sqrt[d]{u} \rceil > u_{\max}$ **do**
  $d \leftarrow d + 1$
end while
$u_r \leftarrow \lceil \sqrt[d]{u} \rceil$
for $j \leftarrow 0$ **to** $d - 1$ **do**
  $\Delta_j \leftarrow \Delta \cdot u_r^{d-1-j}$
  $\boldsymbol{le}^{(j)} \leftarrow \{0, \Delta_j, \cdots, (u_r - 1)\Delta_j\}$
  $\boldsymbol{re}^{(j)} \leftarrow \{\Delta_j - 1, \cdots, u_r\Delta_j - 1\}$
  $\mathsf{isLeft} \leftarrow (j < d - 1)$
  if $j = 0$ **then**
    $[z] \leftarrow \Pi^{\mathrm{ApproxMax1}}_{n_{\mathrm{in}},n,\boldsymbol{le}^{(j)}+lb,\boldsymbol{re}^{(j)}+lb,\mathsf{isLeft}}(\hat{\boldsymbol{x}})$
  else
    $\hat{z} \leftarrow \hat{\Pi}^{\mathrm{S2M}}_{n_{\mathrm{in}}}([z])$
    $[z] \leftarrow [z] + \Pi^{\mathrm{ApproxMax1}}_{n_{\mathrm{in}},n,\boldsymbol{le}^{(j)},\boldsymbol{re}^{(j)},\mathsf{isLeft}}(\hat{\boldsymbol{x}} - \hat{z})$
  end if
end for
return $[z]$

Figure 6: Secure iterative bucket refinement approximate maximum protocol

C Details of Low-Round Secure Element-Wise Nonlinear Protocols

C.1 Design Overview

Low-round secure element-wise nonlinear protocols with reduced computational cost. To enable low-round secure evaluation of element-wise nonlinear functions, we start from FSS-based secure lookup table (LUT). More precisely, we develop a new interval and table-entry construction method for secure interval lookup table (ILUT)-based approximation, and then design secure protocols for nonlinear functions. The key novelty is not merely reducing the table size, but redefining how the input domain is partitioned and how representative outputs are assigned to intervals, so that secure evaluation remains low-round while the cost of FSS-based secure LUT is substantially reduced.

Specifically, we group inputs sharing the same retained top-p significant-bit prefix into one interval and assign them a shared table entry. This design is motivated by the limited effective precision of 16-bit BrainFloat (BF16) Kalamkar et al. (2019). When $p = 8$, the induced interval is aligned with the effective precision of BF16, so inputs in the same interval are already indistinguishable at a precision level widely tolerated in practical neural network inference. As illustrated in Fig. 7, this converts the original LUT into a more compact ILUT. The smaller table is important because the cost of FSS-based secure LUT depends heavily on the table size: reducing the number of table entries lowers the number of AES calls and thereby reduces the computational overhead.

At the protocol level, we instantiate our ILUT-based approximation with different secure ILUT protocols according to the input characteristics. For reciprocal and reciprocal square root, whose input dimensions are relatively small, we use the one-round MIC protocol of Boyle et al. Boyle et al. (2021) to minimize interaction. For exponential and GeLU, which are evaluated over larger input dimensions, we use the two-round secure ILUT protocol of Grotto Storrier et al. (2023) to reduce computational cost.

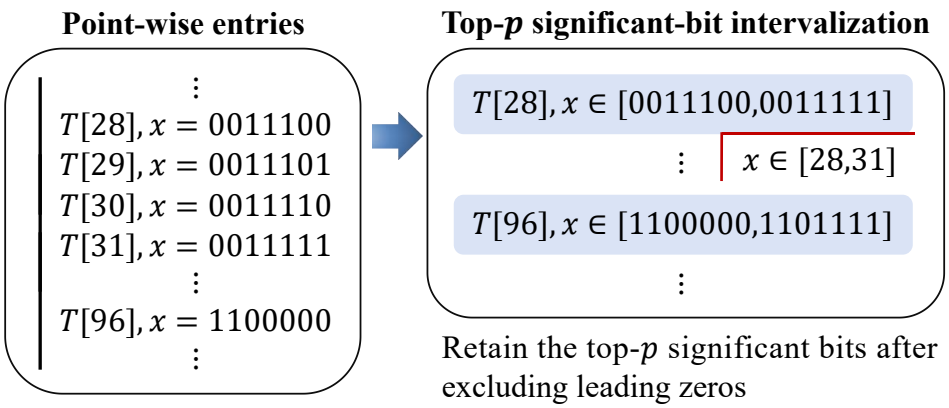


Figure 7: Interval and table-entry construction method

C.2 Secure Protocols

Let the input and output formats be defined by $(n_{\text{in}}, i_{\text{in}}, f_{\text{in}})$ and $(n_{\text{out}}, i_{\text{out}}, f_{\text{out}})$, respectively, where $n = i + f$. Let $[lb, rb]$ denote the valid input range. We first construct an ILUT using the method above, and then evaluate it securely using a low-round ILUT protocol.

Protocol Π^{ILUT1} in Fig. 8 presents a 1-round construction based on Boyle et al.'s MIC protocol Boyle et al. (2021). It directly combines the generated ILUT with Boyle et al.'s 1-round secure MIC, and is particularly suitable for functions with small input dimensions, such as reciprocal and reciprocal square root, where minimizing interaction rounds is the primary objective.

$\Pi^{\text{ILUT1}}_{n_{\text{in}}, i_{\text{in}}, f_{\text{in}}, n_{\text{out}}, i_{\text{out}}, f_{\text{out}}, p, \mathcal{F}, [lb, rb]}(\hat{x})$

1: $n'_{\text{in}}, n'_{\text{out}} \leftarrow i_{\text{in}} + f_{\text{in}}, i_{\text{out}} + f_{\text{out}}$
2: $(\boldsymbol{le}, \boldsymbol{re}), \boldsymbol{T} \leftarrow$
3: $\quad \text{GenILUT}(n'_{\text{in}}, n'_{\text{out}}, f_{\text{in}}, f_{\text{out}}, p, \mathcal{F}, [lb, rb])$
4: $[\boldsymbol{e}] \leftarrow \Pi^{\text{MIC(Boyle)}}_{n'_{\text{in}}, n_{\text{out}}, \boldsymbol{le}, \boldsymbol{re}}(\hat{x})$
5: $[z] \leftarrow \sum_{j=0}^{\text{len}(\boldsymbol{T})-1} [e_j] \cdot T_j$
6: **return** $[z]$

Figure 8: 1-round secure element-wise nonlinear function

Protocol Π^{ILUT2} in Fig. 9 presents a 2-round construction based on Grotto Storrier et al. (2023). Compared with Π^{ILUT1}, it introduces one additional interaction round but reduces the computational cost of secure lookup, making it more suitable for high-dimensional inputs such as exponential and GeLU. Since Grotto's MIC outputs Boolean shares, the protocol first views these shares in the target ring and then performs the sign-correction step before the final multiplication.

$\Pi^{\mathrm{ILUT2}}_{n_{\mathrm{in}},i_{\mathrm{in}},f_{\mathrm{in}},n_{\mathrm{out}},i_{\mathrm{out}},f_{\mathrm{out}},p,\mathcal{F},[lb,rb]}(\hat{x})$

1: $n'_{\mathrm{in}}, n'_{\mathrm{out}} \leftarrow i_{\mathrm{in}} + f_{\mathrm{in}}, i_{\mathrm{out}} + f_{\mathrm{out}}$
2: $(\boldsymbol{le}, \boldsymbol{re}), \boldsymbol{T} \leftarrow$
3: $\quad \mathrm{GenILUT}(n'_{\mathrm{in}}, n'_{\mathrm{out}}, f_{\mathrm{in}}, f_{\mathrm{out}}, p, \mathcal{F}, [lb, rb])$
4: $[\boldsymbol{e}] \leftarrow \Pi^{\mathrm{MIC(Grotto)}}_{n'_{\mathrm{in}},\boldsymbol{le},\boldsymbol{re}}(\hat{x})$
5: View $[\boldsymbol{e}]$ as shares over $\mathbb{Z}_{2^{n_{\mathrm{out}}}}$, denoted by $[\boldsymbol{e}']$
6: $[u] \leftarrow (-1)^b \sum_{j=0}^{\mathrm{len}(\boldsymbol{e}')-1} [e'_j]$
7: $[v] \leftarrow (-1)^b \sum_{j=0}^{\mathrm{len}(\boldsymbol{e}')-1} T_j \cdot [e'_j]$
8: $\hat{u}, \hat{v} \leftarrow \hat{\Pi}^{\mathrm{S2M}}_{n_{\mathrm{out}}}([u]), \hat{\Pi}^{\mathrm{S2M}}_{n_{\mathrm{out}}}([v])$
9: $[z] \leftarrow \Pi^{\mathrm{MUL}}_{n_{\mathrm{out}}}(\hat{u}, \hat{v})$
10: **return** $[z]$

Figure 9: 2-round secure element-wise nonlinear function

D Other Transformer Protocols

The following protocols complete the end-to-end construction in Section 5.

D.1 Secure Embedding Protocol

Figure 10 specifies a privacy-preserving table lookup from the embedding matrix. Given a secret-shared token index vector $\hat{\boldsymbol{x}}$ and a secret-shared embedding matrix $\hat{\boldsymbol{W}} \in \mathbb{Z}_{2^n}^{m \times k}$, the goal is to securely retrieve the corresponding embedding rows without revealing either the queried indices or the embedding parameters.

Our protocol realizes this operation by first converting each secret index into a secret one-hot vector through Π^{Onehot}. It then performs a secure matrix multiplication between the resulting one-hot matrix and the embedding matrix. Since multiplying a one-hot selector with the embedding table is equivalent to row selection, this protocol correctly returns the desired embedding vectors while keeping both the access pattern and the table contents hidden.

$\Pi^{\mathrm{Embedding}}_{n}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{W}})$

1: $[m, k] \leftarrow \mathrm{size}(\hat{\boldsymbol{W}})$
2: $\hat{\boldsymbol{H}} \leftarrow \hat{\Pi}^{\mathrm{Onehot}}_{\lceil \log_2(m) \rceil, n, m}(\hat{\boldsymbol{x}})$
3: $[\boldsymbol{z}] \leftarrow \Pi^{\mathrm{MatMul}}_{n}(\hat{\boldsymbol{H}}, \hat{\boldsymbol{W}})$
4: **return** $[\boldsymbol{z}]$

Figure 10: Secure embedding protocol

D.2 Secure Linear Protocol

Figure 11 specifies an affine transformation of the form $\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{B}$ under fixed-point secret sharing, where $\boldsymbol{W}$ and $\boldsymbol{B}$ denote the weight matrix and bias vector, respectively.

The protocol first invokes Π^{MatMul} to compute the secure matrix multiplication between the input and the weight matrix. Since the multiplication result has doubled scaling in fixed-point representation, it then applies Π^{LocalARS} to restore the target fractional precision. Finally, the bias term is added locally to obtain the output. As this protocol only involves linear operations and a local truncation step, it is one of the most efficient components in secure Transformer inference.

$\Pi^{\mathrm{Linear}}_{n,i,f}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{W}}, [\boldsymbol{B}])$

1: $[\boldsymbol{z}] \leftarrow \Pi^{\mathrm{LocalARS}}_{n,n,f}(\Pi^{\mathrm{MatMul}}_{n}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{W}})) + [\boldsymbol{B}]$
2: **return** $[\boldsymbol{z}]$

Figure 11: Secure linear protocol

D.3 Secure Prefill Protocol

The prefill phase initializes the KV cache by performing a secure forward pass over the entire prompt sequence. For completeness, we describe it as a standalone protocol as Fig. 12. This protocol is executed once to initialize the decoding process. In contrast to decoding, it operates on the full input sequence and computes attention over all positions. The resulting key and value tensors are stored as the initial KV cache and reused in subsequent decoding steps.

$\hat{\Pi}^{\mathrm{Prefill}}_{n,i,f}(\hat{\boldsymbol{x}})$

1: $\hat{\boldsymbol{H}}^{(-1)} \leftarrow \hat{\Pi}^{\mathrm{Embedding}}_{n}(\hat{\boldsymbol{x}})$
2: **for** $\ell = 0$ to $L-1$ **do**
3: $\quad(\hat{\boldsymbol{Q}}^{(\ell)}, \hat{\boldsymbol{K}}^{(\ell)}, \hat{\boldsymbol{V}}^{(\ell)}) \leftarrow \hat{\Pi}^{\mathrm{Linear}}_{n,i,f}(\hat{\boldsymbol{H}}^{(\ell-1)})$
4: $\quad\hat{\boldsymbol{S}}^{(\ell)}_{t} \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,n,f}(\hat{\Pi}^{\mathrm{MatMul}}_{n}(\hat{\boldsymbol{Q}}^{(\ell)}, (\hat{\boldsymbol{K}}^{(\ell)})^{\top}))$
5: $\quad\hat{\boldsymbol{S}}'^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,n,f}(\hat{\boldsymbol{S}}^{(\ell)} \cdot \mathrm{round}(2^{f}/\sqrt{d_k}))$
6: $\quad\hat{\boldsymbol{\alpha}}^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{Softmax}}_{n,i-\lfloor \log_2(\sqrt{d_k})\rfloor,f}(\hat{\boldsymbol{S}}'^{(\ell)} + \boldsymbol{M})$
7: $\quad\hat{\mathbf{O}}^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,n,f}(\hat{\Pi}^{\mathrm{MatMul}}_{n}(\hat{\boldsymbol{\alpha}}^{(\ell)}, \hat{\boldsymbol{V}}^{(\ell)}))$
8: $\quad\hat{\boldsymbol{H}}^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{LayerNorm}}_{n,i,f}(\hat{\boldsymbol{O}}^{(\ell)} + \hat{\boldsymbol{H}}^{(\ell-1)})$
9: $\quad\hat{\boldsymbol{Z}} \leftarrow \hat{\Pi}^{\mathrm{Linear}}_{n,i,f}(\hat{\Pi}^{\mathrm{GeLU}}_{n,i,f}(\hat{\Pi}^{\mathrm{Linear}}_{n,i,f}(\hat{\boldsymbol{H}}^{(\ell)})))$
10: $\quad\hat{\boldsymbol{H}}^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{LayerNorm}}_{n,i,f}(\hat{\boldsymbol{Z}} + \hat{\boldsymbol{H}}^{(\ell)})$
11: **end for**
12: $\hat{\boldsymbol{U}} \leftarrow \hat{\Pi}^{\mathrm{Linear}}_{n,i,f}(\hat{\boldsymbol{H}}^{(L-1)})$
13: **return** $\hat{U}_{\mathrm{len}(\hat{\boldsymbol{x}})-1},\ \{\hat{\boldsymbol{K}}^{(\ell)}, \hat{\boldsymbol{V}}^{(\ell)}\}_{\ell=0}^{L-1}$

Figure 12: Secure prefill protocol

E Detailed Secure Protocols

Figures 13–17 provide the complete pseudocode for the nonlinear operators and decoding workflow summarized in Section 5. They make explicit the fixed-point rescaling, share conversion, and protocol composition omitted from the main text.

Softmax. Figure 13 first reduces the input precision for approximate-maximum computation and subtracts the resulting maximum estimate. It then evaluates the exponential terms with Π^{ILUT2}, evaluates their reciprocal sum with Π^{ILUT1}, and multiplies and rescales the two results to obtain the normalized output.

$\Pi^{\mathrm{Softmax}}_{n,i,f}(\hat{\boldsymbol{x}})$
1: $\hat{\boldsymbol{x}}' \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,i+8,f-8}(\hat{\boldsymbol{x}})$
2: $\hat{\tilde{x}} \leftarrow (\hat{\Pi}^{\mathrm{ApproxMax}}_{i+8,i+f,-2^{i+6},2^{i+6}-1,2^7,2^8}(\hat{\boldsymbol{x}}') + 1) \cdot 2^{f-8}$
3: $\hat{\boldsymbol{t}} \leftarrow \hat{\Pi}^{\mathrm{ILUT2}}_{i+f,i,f,n,i,f,8,\mathcal{F}_{\exp},[-2^{f+4},0]}(\hat{\boldsymbol{x}} - \hat{\tilde{x}})$
4: $\hat{s} \leftarrow \sum_{j=0}^{\mathrm{len}(\hat{\boldsymbol{t}})-1} \hat{t}_j$
5: $\hat{tt} \leftarrow \hat{\Pi}^{\mathrm{ILUT1}}_{i+f,i,f,n,i,f,8,\mathcal{F}_{\mathrm{recip}},[2^{f-1},2^f \cdot \mathrm{len}(\hat{\boldsymbol{x}})]}(\hat{s})$
6: $[z] \leftarrow \Pi^{\mathrm{LocalARS}}_{n,n,f}(\Pi^{\mathrm{MUL}}_n(\hat{\boldsymbol{t}}, \hat{tt}))$
7: **return** $[z]$

Figure 13: Secure Softmax protocol

GeLU. Figure 14 evaluates GeLU as the sum of a ReLU term and a bounded correction term. Grotto's MIC protocol identifies the interval containing the input, while a secure comparison determines its sign; after share conversion, the protocol combines the corresponding lookup-table value with the selected ReLU output.

$\Pi^{\mathrm{GeLU}}_{n,i,f}(\hat{x})$
1: $(\boldsymbol{le}, \boldsymbol{re}), \boldsymbol{T} \leftarrow$
2: GenILUT$(i+f, i+f, f, f, 8, \mathcal{F}_\delta, [-2^{2+f}, 2^{2+f}])$
3: $[\boldsymbol{e}] \leftarrow \Pi^{\mathrm{MIC(Grotto)}}_{i+f,\boldsymbol{le},\boldsymbol{re}}(\hat{x})$
4: $[s] \leftarrow \Pi^{\mathrm{Greater}}_{i+f}(\hat{x}, 0)$
5: View $[\boldsymbol{e}]$ as shares over $\mathbb{Z}_{2^n}$, denoted by $[\boldsymbol{e}']$
6: $[u] \leftarrow (-1)^b \sum_{j=0}^{\mathrm{len}(\boldsymbol{e}')-1} ([e'_j] \bmod 2)$
7: $[v] \leftarrow (-1)^b \sum_{j=0}^{\mathrm{len}(\boldsymbol{e}')-1} T_j \cdot ([e'_j] \bmod 2)$
8: $\hat{s}, \hat{u}, \hat{v} \leftarrow \hat{\Pi}^{\mathrm{S2M}}_1([s]), \hat{\Pi}^{\mathrm{S2M}}_n([u]), \hat{\Pi}^{\mathrm{S2M}}_n([v])$
9: $[z] \leftarrow \Pi^{\mathrm{Select}}_n(\hat{s}, \hat{x}) + \Pi^{\mathrm{MUL}}_n(\hat{u}, \hat{v})$
10: **return** $[\boldsymbol{z}]$

Figure 14: Secure GeLU protocol

LayerNorm. Figure 15 computes the mean and variance under fixed-point secret sharing, converts the variance into masked shares, and evaluates its reciprocal square root with Π^{ILUT1}. The final two multiplications normalize the input and apply the learnable affine parameters.

$\Pi^{\mathrm{LayerNorm}}_{n,i,f}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\gamma}}, [\boldsymbol{\beta}])$
1: $m \leftarrow \mathrm{len}(\boldsymbol{x})$
2: $\hat{\mu} \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,n,f}(\mathrm{round}(2^f/m) \cdot \sum_{j=0}^{m-1} \hat{\boldsymbol{x}}_j)$
3: $[\boldsymbol{t}] \leftarrow \Pi^{\mathrm{LocalARS}}_{n,n,f}(\Pi^{\mathrm{MUL}}_n(\hat{\boldsymbol{x}} - \hat{\mu}, \hat{\boldsymbol{x}} - \hat{\mu}))$
4: $[\sigma^2] \leftarrow \Pi^{\mathrm{LocalARS}}_{n,n,f}(\mathrm{round}(2^f/m) \cdot \sum_{j=0}^{m-1} [t_j])$
5: $\hat{\sigma^2} \leftarrow \hat{\Pi}^{\mathrm{S2M}}_n([\sigma^2])$
6: $\hat{tt} \leftarrow \hat{\Pi}^{\mathrm{ILUT1}}_{i+f,i,f,n,i,f,\mathcal{F}_{\mathrm{rsqrt}},[1,2^{i+f-1}]}(\hat{\sigma^2})$
7: $\hat{\boldsymbol{z}} \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,n,f}(\hat{\Pi}^{\mathrm{MUL}}_n(\hat{\boldsymbol{x}} - \hat{\mu}, \hat{tt}))$
8: $[\boldsymbol{z}] \leftarrow \Pi^{\mathrm{LocalARS}}_{n,n,f}(\Pi^{\mathrm{MUL}}_n(\hat{\boldsymbol{z}}, \hat{\boldsymbol{\gamma}})) + [\boldsymbol{\beta}]$
9: **return** $[\boldsymbol{z}]$

Figure 15: Secure LayerNorm protocol

One-Step Decoding. Figure 16 composes the secure Transformer operators for one decoding step. It embeds the current token, locally appends the newly computed keys and values to each layer's cache, performs attention and feed-forward computation, and returns both the output logits and the updated KV cache.

$\hat{\Pi}^{\mathrm{DecodeStep}}_{n,i,f}(\hat{x}_t, \{\hat{\boldsymbol{K}}^{(\ell)}_{0:t-1}, \hat{\boldsymbol{V}}^{(\ell)}_{0:t-1}\}_{\ell=0}^{L-1})$

$\hat{\boldsymbol{h}}_t^{(-1)} \leftarrow \hat{\Pi}^{\mathrm{Embedding}}_{n}(\hat{x}_t)$
for $\ell = 0$ to $L-1$ **do**
  $(\hat{\boldsymbol{q}}_t^{(\ell)}, \hat{\boldsymbol{k}}_t^{(\ell)}, \hat{\boldsymbol{v}}_t^{(\ell)}) \leftarrow \hat{\Pi}^{\mathrm{Linear}}_{n,i,f}(\hat{\mathbf{h}}_t^{(\ell-1)})$
  $\hat{\boldsymbol{K}}^{(\ell)}_{0:t} \leftarrow \mathrm{Concat}(\hat{\boldsymbol{K}}^{(\ell)}_{0:t-1}, \hat{\boldsymbol{k}}_t^{(\ell)})$
  $\hat{\boldsymbol{V}}^{(\ell)}_{0:t} \leftarrow \mathrm{Concat}(\hat{\boldsymbol{V}}^{(\ell)}_{0:t-1}, \hat{\boldsymbol{v}}_t^{(\ell)})$
  $\hat{\boldsymbol{s}}_t^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,n,f}(\hat{\Pi}^{\mathrm{MatMul}}_{n}(\hat{\boldsymbol{q}}_t^{(\ell)}, (\hat{\boldsymbol{K}}^{(\ell)}_{0:t})^{\top}))$
  $\hat{\boldsymbol{s}}'^{(\ell)}_t \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,n,f}(\hat{\boldsymbol{s}}_t^{(\ell)} \cdot \mathrm{round}(2^f/\sqrt{d_k}))$
  $\hat{\boldsymbol{\alpha}}_t^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{Softmax}}_{n,i-\lfloor \log_2(\sqrt{d_k}) \rfloor,f}(\hat{\boldsymbol{s}}'^{(\ell)}_t)$
  $\hat{\mathbf{o}}_t^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{LocalARS}}_{n,n,f}(\hat{\Pi}^{\mathrm{MatMul}}_{n}(\hat{\boldsymbol{\alpha}}_t^{(\ell)}, \hat{\boldsymbol{V}}^{(\ell)}_{0:t}))$
  $\hat{\boldsymbol{h}}_t^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{LayerNorm}}_{n,i,f}(\hat{\boldsymbol{o}}_t^{(\ell)} + \hat{\boldsymbol{h}}_t^{(\ell-1)})$
  $\hat{\boldsymbol{z}} \leftarrow \hat{\Pi}^{\mathrm{Linear}}_{n,i,f}(\hat{\Pi}^{\mathrm{GeLU}}_{n,i,f}(\hat{\Pi}^{\mathrm{Linear}}_{n,i,f}(\hat{\boldsymbol{h}}_t^{(\ell)})))$
  $\hat{\boldsymbol{h}}_t^{(\ell)} \leftarrow \hat{\Pi}^{\mathrm{LayerNorm}}_{n,i,f}(\hat{\boldsymbol{z}} + \hat{\boldsymbol{h}}_t^{(\ell)})$
end for
$\hat{\boldsymbol{u}}_t \leftarrow \hat{\Pi}^{\mathrm{Linear}}_{n,i,f}(\hat{\boldsymbol{h}}_t^{(L-1)})$
return $\hat{\boldsymbol{u}}_t,\ \{\hat{\boldsymbol{K}}^{(\ell)}_{0:t}, \hat{\boldsymbol{V}}^{(\ell)}_{0:t}\}_{\ell=0}^{L-1}$

Figure 16: Secure one-step decoding protocol

Autoregressive Decoding. Figure 17 composes one Prefill invocation with repeated calls to the one-step decoding protocol. At each step, the logits are revealed to the user for local token selection; the selected token is then masked and returned to the SMPC parties, which reuse the updated KV cache rather than recomputing the prefix.

$\Pi^{\mathrm{Decoding}}_{n,i,f}(\hat{\boldsymbol{x}}, T)$

$(\hat{\boldsymbol{u}}_m, \{\hat{\boldsymbol{K}}^{(\ell)}_{0:m}, \hat{\boldsymbol{V}}^{(\ell)}_{0:m}\}_{\ell=0}^{L-1}) \leftarrow \hat{\Pi}^{\mathrm{Prefill}}_{n,i,f}(\hat{\boldsymbol{x}})$
Reveal $\boldsymbol{u}_m$ to the user
for $t = m+1$ to T **do**
  User selects x_t based on $\boldsymbol{u}_{t-1}$, then mask $\hat{x}_t$ and send to SMPC parties
  SMPC parties call the protocol $\hat{\Pi}^{\mathrm{DecodeStep}}_{n,i,f}$ and get $(\hat{\boldsymbol{u}}_t, \{\hat{\boldsymbol{K}}^{(\ell)}_{0:t}, \hat{\boldsymbol{V}}^{(\ell)}_{0:t}\}_{\ell=0}^{L-1})$
  SMPC parties reveal $\boldsymbol{u}_t$ to the user
end for

Figure 17: Secure autoregressive decoding protocol

F Additional Evaluation Results

This section supplements the evaluation in Section 6 with approximation-error measurements and a decoder-block breakdown. Unless stated otherwise, all experiments use the models, tensor shapes, fixed-point formats, and network settings described in the main paper.

F.1 Approximation Accuracy

Table 7 compares the mean absolute error (MAE) and maximum error of RESAD and SIGMA for Softmax, GeLU and LayerNorm over the evaluated input ranges. RESAD has a higher Softmax

MAE but a lower Softmax maximum error, reduces both GeLU error metrics, and matches SIGMA for LayerNorm. Thus, the approximation errors of the two schemes remain of the same order of magnitude across all three operators.

Table 7: Approximation errors of secure Transformer operators

Operator	Scheme	MAE	Maximum Error
Softmax	SIGMA	$\mathbf{1.19 \times 10^{-4}}$	2.64×10^{-2}
	Ours	2.39×10^{-4}	$\mathbf{6.83 \times 10^{-3}}$
GeLU	SIGMA	3.55×10^{-4}	7.31×10^{-3}
	Ours	$\mathbf{1.40 \times 10^{-4}}$	$\mathbf{1.38 \times 10^{-3}}$
LayerNorm	SIGMA	1.05×10^{-3}	1.32×10^{-2}
	Ours	1.05×10^{-3}	1.32×10^{-2}

F.2 Decoder-Block Breakdown

Table 8 isolates the effect of our protocols within one GPT-2 Small decoder block at the 512th token. RESAD reduces the round count from 69 to 26 and lowers WAN online and total runtime by 60.1% and 31.9%, respectively.

Table 8: Costs of a secure GPT-2 Small decoder block at the 512th token

Network	Scheme	Rounds	Comm. (MB)		Time (ms)	
			Online	Total	Online	Total
LAN	SIGMA	69	1.49	**8.87**	210.24	363.74
	Ours	**26**	**0.78**	9.58	**176.29**	**353.62**
WAN	SIGMA	69	1.49	**8.87**	2994.19	4796.76
	Ours	**26**	**0.78**	9.58	**1193.69**	**3265.70**

G Security Analysis

By the composition theorem (Canetti, 2000), if each sub-protocol securely realizes its corresponding ideal functionality, then any protocol obtained by sequentially or concurrently composing these sub-protocols also securely realizes the corresponding composed functionality. Our protocol is constructed by invoking secure building blocks. Therefore, in the static semi-honest model, the view of each party can be simulated from its input and output, and the composed protocol reveals no information beyond the prescribed outputs.

H Clarification on SHAFT

During our examination of the experimental results reported by SHAFT (Kei & Chow, 2025), we observed an inconsistency between the bandwidth units stated in its evaluation and those implied by its runtime calculations. In Table VII of the SHAFT paper (Kei & Chow, 2025), SHAFT directly quotes the runtime results of the compared schemes from their original papers and labels their network settings as a LAN with $1\,\mathrm{Gbps}$ bandwidth and $0.5\,\mathrm{ms}$ latency, and a WAN with $400\,\mathrm{Mbps}$ bandwidth and $4\,\mathrm{ms}$ latency. These settings are consistent with those explicitly specified in the corresponding original papers. However, SHAFT's own reported runtimes and the calculation provided in its appendix indicate that its numerical bandwidth values were interpreted in bytes per second rather than bits per second. For example, for BERT-base, SHAFT reports a computation time of $6.93\,\mathrm{s}$, a communication volume of $10.46\,\mathrm{GB}$, and 1495 communication rounds, and calculates the LAN runtime as

$$6.93 + \frac{2 \times 10.46}{1} + \frac{1495 \times 0.5}{1000} = 28.60\,\mathrm{s}.$$

This result is obtained only when the bandwidth value 1 is interpreted as $1\,\mathrm{GB/s}$. Under the stated $1\,\mathrm{Gbps}$ setting, converting the communication volume from bytes to bits introduces an additional factor of eight. Similarly, SHAFT's WAN runtimes are consistent with a bandwidth of

$400\,\mathrm{MB/s}$, rather than $400\,\mathrm{Mbps}$. Consequently, the baseline runtimes directly quoted from the original papers correspond to $1\,\mathrm{Gbps}/400\,\mathrm{Mbps}$, whereas SHAFT's own runtimes were calculated using $1\,\mathrm{GB/s}/400\,\mathrm{MB/s}$, resulting in an approximately eightfold difference in the effective bandwidth scale. To ensure a consistent comparison, we recalculate SHAFT's runtimes using bandwidth values expressed uniformly in bits per second throughout our evaluation.